An extensive appraisal of weight-sharing on the NAS-Bench-101 benchmark

Aloïs Pourchot, Kévin Bailly, Alexis Ducarouge, Olivier Sigaud

An Extensive Appraisal of Weight-Sharing on the NAS-Bench-101 Benchmark

Aloïs Pourchot[a,b,*], Kévin Bailly[a], Alexis Ducarouge[b], Olivier Sigaud[a]

[a] *Sorbonne Université, CNRS UMR 7222, Institut des Systèmes Intelligents et de Robotique, F-75005 Paris, France*
[b] *Gleamer, 117 Quai de Valmy, 75010 Paris, France*

Abstract

Weight-sharing (WS) has recently emerged as a paradigm to accelerate the automated search for efficient neural architectures, a process dubbed Neural Architecture Search (NAS). By using and training the same set of weights for the whole search space, WS allows for the quick evaluation of millions of architectures, where classical NAS approaches require lengthy individual trainings. Although very appealing, WS is not without drawbacks and several works have started to question its capabilities on small hand-crafted benchmarks. In this paper, we take advantage of the NAS-Bench-101 dataset to challenge the efficiency of a uniform-sampling based WS variant on several representative search spaces. After reviewing previous studies on WS and highlighting several of their shortcomings, we introduce our own experimental setup, from which we extract several good practices that one should keep in mind when evaluating WS. With our experiments we first establish that, given the correct evaluation procedure, WS is able to produce accuracy

*Corresponding author
Email address: alois.pourchot@gmail.com (Aloïs Pourchot)

scores decently correlated with standalone ones. We then provide evidence that on some search spaces, this WS variant is able to rapidly find better than random architectures, whilst it is equivalent or sometimes even worse than a baseline random search on others, as we find that given the same budget, the probability of superiority of an architecture found using WS over an architecture found through random search can vary between 7% and 78% depending on the search space. We present evidence that the search space itself has an intricate effect on the capabilities of WS and can bias weight-sharing towards certain architectural patterns with no clear accuracy advantage. We conclude that the impact of WS is heavily search-space dependent and difficult to anticipate for a given problem.
Keywords: Deep convolutional neural networks, neural architecture search (NAS), weight-sharing (WS), computer vision.

---

1. Introduction

Using deep neural networks (DNNs) has led to numerous breakthroughs on many hard machine learning tasks, such as object detection and recognition or natural language processing [1]. In the last years, a paradigm shift was observed, from hand-designing features that can be fed to a machine learning algorithm, to hand-designing neural architectures that can extract those features automatically. However, arranging DNNs is itself time-consuming, requires a lot of expertise and remains very domain-dependent. A promising approach is to automatically design them, a process referred to as Neural Architecture Search (NAS) [2, 3].

Regrettably, because of expensive training requirements, evaluating a sin-

gle DNN architecture can take days to weeks. In turn, original NAS approaches [4, 5, 6] required thousands of GPU days worth of computing, only to find conformations slightly better than expert-designed ones. In light of this concern, many methods have been explored that could drastically cut the resources required to perform NAS, and today's literature is blooming with approaches requiring less than a day of computations [7, 8, 9, 10]

Most of these efficient methods rely on a computational trick called weight-sharing (WS). Although weight-sharing can refer to various methods as it was coined independently several times with different meanings [11, 7, 12, 13], we refer here to the approach first popularized by [11] and [7], where authors propose to reuse sets of weights from previously trained networks to bootstrap the training of new ones, rather than training each newly chosen architecture from scratch. Pham et al. [7] further notice that in specific search spaces, each network can be seen as a sub-graph of a larger graph, which is widely referred to as the "super-net" in the NAS community. With WS, one can train the whole search space at once, by using a single set of weights, from which the parameters of each single possible model can be extracted.

Despite a growing literature, the effect of WS on the performance of NAS is still poorly understood. A particular concern is the quality of the scores obtained with the super-net. Employing WS implies substituting metrics obtained after standalone training with metrics derived from the shared set of parameters. Both quantities thus need to be correlated: if networks with excellent standalone performance are under-evaluated by the super-net or vice-versa, the process could be pointless and even detrimental.

A possible reason for a lack of studies on the matter is the cost required by the training of a proper amount of architectures. As described in Section 3, several works mitigate this issue by assessing the correlations between evaluations of the super-net and true evaluations in reduced settings, either evaluating few architectures or studying a drastically reduced search space.

In this paper, we leverage the architecture evaluations of the NAS-Bench-101 [14] dataset to scrutinize the correlations offered by a simple WS variant and investigate whether it can improve the efficiency of baseline NAS algorithms. We perform this comparison in a realistic setup in which we account for all the computational costs incurred by WS. To strengthen the results of our different analyses, we consider seven different search spaces with specific structural properties and reproduce this analysis on each of them. Besides, to grant the results clear statistical significance, we compare the different methods with an appropriate number of runs given a pre-specified effect size, totaling around 25 WS training for each search space. To our knowledge, this study is the first to consider analysing WS on such a wide variety of search spaces, using a fair comparison protocol that explores all sources of computational cost, and with strong enough statistical power to draw meaningful conclusions.

We summarize the key take-aways of our experimental setup in a list of several good practices, which we believe are essential to accurately evaluate WS. They are the following: evaluating WS using a wide variety of search spaces, avoiding the different sources of biases that could come up during the estimation of correlations, visualizing said correlations, properly assessing computation duration, and guaranteeing statistical significance of the NAS

results.

Based on these observations and the resulting good practices, our experimental results show that: (i) with the correct methodology, one can get decent Spearman's rank correlations between super-net proxy evaluations and real evaluations on several search spaces containing hundred thousands of architectures, which can range between 0.46 and 0.71 depending on the search space; (ii) WS is often able to quickly spot better than random architectures, but is sometimes on par or even worse than a random search baseline, as the probability of superiority of WS over random search under a one-shot setting can vary between 7% and 78% depending on the search space; (iii) global correlations evidenced in (i) are not the limiting factor of uniform-sampling based WS, as search spaces offering better global correlation do not always offer better WS performances; (iv) the search-space can bias the super-net training and the resulting evaluations, making WS fairly unreliable.

2. Background

In this Section, we introduce the foundations of NAS optimization before describing the basic WS approach and some improvements to the standard super-net training that have been suggested in the literature. We then rapidly describe the NAS-Bench-101 dataset.

2.1. NAS Optimization

Given a set of possible architectures $\mathcal{A}$, each of them coming with a set of possible weights $\mathcal{W}(\mathcal{A})$, the goal of NAS can be formulated as a bi-level

optimization process [15], which takes the following generic form:

$$\text{Find } A \in \arg\min_{A \in \mathcal{A}} \Gamma(A), \text{ where } \Gamma(A) = \inf \left\{ F(A, w^\star), w^\star \in \arg\min_{w \in \mathcal{W}} f(A, w) \right\}. \tag{1}$$

In bi-level optimization problems, the evaluation of the variable of interest depends on a second variable (here A and w respectively) that is constrained to be optimal with respect to another optimization problem. F and A are called the outer objective and variable, whereas f and w are the inner objective and variable. In the typical context of machine learning, f would be the loss of a model on a training dataset, whereas F would be the loss of the same model on a held-out validation dataset. During training, the model only has access to the training dataset to update its parameters in order to minimize f. However, we would rather like the architecture of the model to be selected so as to improve the model generalization, that is to decrease the loss of the model on unseen data, represented by the validation dataset. A bi-level optimization typically encountered in machine learning is the hyper-parameter optimization problem, of which NAS is a particular instance.

2.1.1. Dynamic Formulation of the Constraint

Under its general form, the problem presented in (1) creates constraints on the inner variable w that cannot be explicitly formulated in terms of A. Several optimal values might also exist. As a result, $w^\star \in \arg\min_{w \in \mathcal{W}} f(A, w)$ cannot be expressed as a function of A, an undesirable property when searching for practical resolution algorithms. To circumvent this difficulty, the

assumption is often implicitly made that each possible value of $w^\star$ corresponds to the same value of the outer objective F. It is also assumed that at least one set of optimal weights can be reached through an iterative process, which dynamics can be described analytically in terms of A. In practice, (1) is replaced by:

$$\text{Find } A \in \arg\min_{A \in \mathcal{A}} F(A, w_T(A)), \tag{2}$$
$$\text{u.c.} \left\{ \begin{array}{rcl} w_0 & = & \Phi_0(A, \epsilon) \\ w_t & = & \Phi_t(w_{t-1}, A, \epsilon)\, t = 1, \ldots, T, \end{array} \right\}$$

where the initial weights w_0 are initialized in a deterministic way or through some random process, characterized by a random variable ϵ. Each w_t is obtained by applying the process Φ_t to w_{t-1} and the current architecture A. Typically, Φ_t takes the form of a stochastic gradient update on a loss l computed using a mini-batch of data sampled from a training set:

$$\Phi_t(w_{t-1}, A, \epsilon) = w_{t-1} - \alpha \nabla_w \mathbb{E}_{x,y \sim \mathcal{D}_t}[l(A, w, x, y)]. \tag{3}$$

If the inner optimization behaves well, then as $T \to \infty$ the solutions of (2) should get closer to the solutions of (1). Franceschi et al. [16] show that this is true under reasonable assumptions.

2.1.2. Continuous and Stochastic Relaxations

In a NAS problem, the set of possible architectures $\mathcal{A}$ is usually discrete. Some approaches directly solve the outer optimization problem using this discrete representation, making use of either genetic algorithms [4, 17], reinforcement learning algorithms [18] or hyper-parameter optimization algorithms [19]. When an architecture A is to be evaluated, the corresponding

$w_T(A)$ is computed and the resulting fitness of the individual is calculated as $F(A, w_T(A))$.

To work with continuous rather than discrete variable, other NAS approaches rely on some form of relaxation of the objective in (2). In [8], the authors introduce a continuous relaxation of the search space itself: an architecture is considered as a weighted average of all possible architectures. Given a set of elementary operations $\mathcal{O} = (o_i)_i$, each choice of such operation o_j is replaced by a weighted average operation: $\tilde{o_j} = \sum_i \lambda_{i,j} o_i$. Typically, those elementary operations take the form of different convolution kernels, pooling kernels or skip connections. This makes the problem differentiable with respect to the architecture parameters λ. The problem described in (2) becomes:

$$\begin{aligned} &\text{Find } \lambda \in \arg\min_{\lambda \in \Lambda} F(A(\lambda), w_T), \\ &\text{u.c.} \left\{ \begin{array}{rcl} w_0 & = & \Phi_0(A(\lambda), \epsilon) \\ w_t & = & \Phi_t(w_{t-1}, A(\lambda), \epsilon), t = 1, \ldots, T. \end{array} \right\} \end{aligned} \tag{4}$$

Although very straightforward, this approach is not resource-efficient, as all the allowed elementary operations of each node appear in the computational graph. Several works have since improved in various ways the original idea [20, 21].

Numerous other authors [5, 9, 10] rather consider a stochastic relaxation of the problem. Instead of working with $\mathcal{A}$ directly, a family of probability distributions $\mathcal{P} = \{P_\theta : \theta \in \Theta \ \subseteq \mathbb{R}^{n_\theta}\}$ is introduced over the set of architectures [22]. It is assumed that each $P_\theta \in \mathcal{P}$ has a density function p_θ which is differentiable with respect to θ. Working with this family, the outer objective

in (2) is replaced by:

$$\text{Find } \theta \in \arg\min_{\theta \in \mathbb{R}^{n_\theta}} \mathbb{E}_{A \sim P_\theta}\big\{F(A, w_T)\big\}. \tag{5}$$

The idea behind this relaxation is that optimality is reached if P_θ concentrates all its mass in a Dirac centered at the optimal architecture. This results in the following optimization problem:

$$\text{Find } \theta \in \arg\min_{\theta \in \mathbb{R}^{n_\theta}} \mathbb{E}_{A \sim P_\theta}\big\{F(A, w_T(A))\big\} \tag{6}$$
$$\text{where } \left\{ \begin{array}{rcl} w_0(A) & = & \Phi_0(A, \epsilon) \\ w_t(A) & = & \Phi_t(w_{t-1}, A, \epsilon), t = 1, \ldots, T. \end{array} \right\}$$

The constraints of the original formulation disappear (because A is now a dummy variable in the expectation), which renders θ completely independent from the sequence of weights encountered during the inner optimization and makes the objective trivially differentiable in θ. In this case, the resulting gradient is a REINFORCE like type gradient. Notice that this black-box approach is similar to the gradient-free methods used in the discrete case, and is in particular very akin to evolutionary algorithms such as evolution strategies [23].

2.2. Weight-Sharing

Weight-sharing refers to combining the weights of all the architectures of a search space into a single super-net. To access a model and its weights, one only needs to activate the corresponding computational sub-graph. The shared parameters are learned by successively activating different parts of the super-net and performing the standard forward-backward propagation algorithm using mini-batches of training data.

2.2.1. WS Optimization

WS approximates the solution to a NAS problem by successive iterations of two steps:

$$\text{Find} \quad W \in \arg\min_{W \in \mathbb{R}^N} \Phi(\mathcal{A}, f, W), \tag{7}$$

$$\text{Find} \quad A \in \arg\min_{A \in \mathcal{A}} F(A, W), \tag{8}$$

where $\mathcal{A}$ is the set of possible architectures, W the weights of the super-net, F is the outer objective (usually a validation loss), f is the inner objective (usually a training loss) and Φ is a function of the inner objective and the search space which dictates how to optimize W. The loss Φ is usually expressed as an expectation of the inner objective over a distribution P_θ of architectures:

$$\Phi(\mathcal{A}, f, W) = \mathbb{E}_{A \sim P_\theta(\mathcal{A})}\{f(A, W)\}. \tag{9}$$

Different approaches combine both phases in different ways. Most alternate between the two, with mini-batches of data respectively coming from training and validation sets [8, 9, 10, 7]. In this work, we first train a super-net until convergence, and then use it to select possibly good architectures. We take as baseline the work of [24], where models are sampled uniformly from the set of all possible architectures and the super-net is updated in accordance. This paradigm facilitates the analysis of correlations, as methods training both weights and architectures together induce a bias towards architectures with good early evaluations. Besides, [24] report better performance when exploiting their trained super-net to perform NAS.

2.2.2. Enhancing Weight-Sharing Correlations

Several tricks and WS training variants have been introduced in the literature to improve the correlations granted by the super-net. We list several unrelated approaches here, which we explore in our experiments in Section 4.2 and 5.1.

During evaluation, it is possible to directly exploit the whole super-net and perform a standard forward pass on the impending data whilst activating the graph corresponding to the evaluated net. However, several works report the benefit of adapting the statistics of the inherited batch normalization layers [25, 24].

The weights of the super-net W are updated through gradient descent with respect to the objective in (7), using the formulation of Φ described in (9). The resulting gradient takes the form of an expectation over the distribution P_θ:

$$\nabla_W \Phi(A, f, W) = \mathbb{E}_{A \sim P_\theta(\mathcal{A})}\{\nabla_W f(A, W)\}. \tag{10}$$

This expectation is approximated by an empirical average, using random architectures sampled following $P_\theta(\mathcal{A})$. However, in practice Guo [24] only use a single architecture to estimate the expectation. Although this process is unbiased, it results in high variance updates of W. Decreasing this variance by sampling more models could improve the super-net optimization, at a higher computational cost.

In [26], the authors propose to not only share the weights of the basic operations between architectures, but to further merge the weights of all basic operations at a given node into a single set of parameters. For instance, if two basic operations were a $x \times x$ convolution and a $y \times y$ convolution with

$x > y$, instead of representing each operation with its own set of kernels, one could use a single set of size $x \times x$, and apply the $y \times y$ convolution by extracting the sub-kernels of size $y \times y$ from the the shared set.

Authors of [27] identify in their work a bias towards architectures with fewer parameters, as they get trained faster. They propose to correct this bias by sampling architectures pro-rata to their number of parameters, sampling complex architectures more often.

3. Related Work

In this section we describe several works trying to measure the efficiency of weight-sharing.

In [25] the authors train a super-net on a search space of their own. Path dropout is applied during training to randomly cut some portions out of the super-net. A random search is then used to find a good architecture. To validate the use of the super-net as a proxy, $20,000$ architectures are sampled from the chosen search space and evaluated with the resulting super-net. This set is then partitioned into several bins based on the obtained proxy scores. For each bin, 4 architectures are sampled and trained from scratch for a small number of epochs (around 10% of the length of baseline training) before being evaluated. The authors note visually satisfying correlations between the two proxies, but do not report any metrics. For computational reasons, correlations with full budget standalone accuracy are not reported. Moreover, because the few models evaluated are evenly spread across the range of possible proxy accuracies, the produced appealing correlations plots might not be representative of the whole search space.

In [28], Sciuto et al. quantify the impact of WS on NAS-Bench-101, and on a small language modeling task. They find that a simple random search baseline is competitive with and often outperforms several NAS algorithms exploiting WS such as DARTS [8], ENAS [7] and NAO [29], which furthermore display high variance in their results. They report poor correlations between the ranks obtained using a super-net and with standalone evaluations on the considered search spaces. We note however that the used algorithms were not specifically designed to produce good correlations at the end of training, but rather exploit them to rapidly converge to seemingly good architectures.

Zhang et al. [30] explore another small search space of 64 architectures dedicated to computer vision. They train several super-nets using different seeds, and report high variance in the relative rankings of the architectures obtained with WS. They notice that during super-net training, strong interactions exist between architectures, as updates in some models can either improve or deteriorate the performance of others. They reach correct correlations with standalone rankings, albeit the important variance seems to hinder the practical implications of the super-net. They propose several approaches to reduce the amount of WS between architectures, such as fine-tuning parts of the super-net before evaluation or grouping architectures into different sets according to different strategies.

In [31], Chu et al. argue that WS is limited by uneven sampling of individual weights throughout the learning process. Although they are seen equally often on average, some might locally be over-represented due to chance, effectively biasing the weights of the super-net. To prevent this, they propose to

average the gradient updates of the shared parameters over n samples, chosen such that each of the n elementary operations of the super-net appears exactly once. They combine their super-net trained with the aforementioned strategy with a multi-objective genetic algorithm, to build a Pareto front of accurate architectures with adequate numbers of parameters and multiply-add operations. To justify their approach, they sample 13 models equally distant from the found Pareto front and compare the accuracy obtained with WS against standalone ones, and reveal that rankings are well preserved. However, details regarding this experiment are lacking and it is likely that the result does not hold for the whole search space.

Luo et al. [27] also note a strong variance in the results of a few NAS algorithms exploiting WS, and impute the poor results to meager correlations, which they illustrate using 50 randomly sampled architectures. After identifying several factors that could hinder performances, such as short training times and bias towards simple architectures, they propose straightforward solutions for each of them, and eventually demonstrate improved performances.

In [32], the authors benchmark 8 NAS methods, 6 of which are based on WS, on 5 different datasets. Their careful analysis reveals that many NAS algorithms have trouble outperforming a random search. They further argue that the use of training tricks in the evaluation protocol have a greater influence over the final performance than the NAS algorithm itself. Besides, they note that search spaces typically used in NAS have a very narrow accuracy range and are thus already tuned to the considered tasks.

Zela et al. [33] use NAS-Bench-101 to evaluate NAS algorithms exploiting WS. However, because the authors choose to study DARTS variants, they

create their own search spaces to perform evaluations, whereas we directly use the whole NAS-Bench-101 search space. In one of their experiments, they report the evolution during training of the correlations between evaluations obtained using the super-net, and evaluations queried from the NAS-Bench-101 dataset. They report poor or nonexistent correlations for most algorithms, which seems to contradict our findings.

Posterior to an early version of our work [34], Zhang et al. [35] perform experiments with an intent similar to that of our study. Instead of considering several sub-spaces of NAS-Bench-101, the authors rely on different search-spaces typically found in the NAS literature. On NAS-Bench-101, the authors consider a relatively small sub search-space, introduced by [33], which contains around $42,000$ architectures, around 10 times less than the number of available unique architectures. For each search space, they train several super-nets using the same uniform-sampling variant of WS [24]. Their results suggest that it is possible to reach correct correlations, but that there is an important variance with respect to the search space considered. This variance of behavior is more preeminent in their work, given the greater diversity of search spaces considered. Because the search space that they consider are not all based on available NAS benchmarks, some of those correlations are obtained by training only a few tens of architectures. They additionally reveal that on some search spaces uniform WS is biased towards certain operators, and that although WS cannot be used to accurately select top architectures, it is paradoxically quite capable of finding the worst ones. Unlike this work, the authors conclude that the positive effect of WS on NAS is clear cut and positive. We explain this discrepancy by the fact that they

do not account for super-net training times when comparing against random search.

For clarity purposes, we gather in Table 1 a quick summary of each work described in this section and their conclusions on super-net correlations and whether WS can improve NAS consistently. We embed each paper using five dimensions: the approximate size of the considered search spaces, the number of studied search spaces, the number of architecture evaluations performed, whether the architecture evaluations are biased and whether the WS variant used uniform sampling. We additionally highlight in red what we think are shortcomings of the different studies, which we briefly describe here. The authors of [30] consider an unrealistic search space of merely 64 architectures. All studies except [33, 35] and ours consider less than three search spaces, which we think is too narrow to get an understanding of the various behaviors of WS. Almost all authors evaluate below 1,000 architecture when analysing correlations, resulting in limited statistical significance. The authors of [25, 31] perform biased evaluations which are likely to overestimate the quality of the correlations offered by WS. On the contrary, authors of [28] report their results for WS based NAS algorithms that do not use uniform sampling and are therefore likely to underestimate the quality of the correlations offered by WS.

All in all, the authors of [25, 30, 31, 35] report reaching correct correlations whilst the authors of [28, 27, 33] report poor correlations, both sides coming with various shortcomings in the methodology. Regarding NAS performances when exploiting WS, authors of [28, 32], suggest that when properly evaluated WS-based NAS are no better than random search, whilst

AUTHORS	SPACE SIZE	#SPACES	#EVALS	UNBIASED EVALS	UNIFORM WS	GOOD CORRS.	WS ↗
[25]	$\sim 10^9$	1	$\sim 10^2$	×	✓	✓	✓
[28]	$\sim 10^5$	2	200	✓	×	×	×
[30]	$\sim 10^1$	1	64	✓	✓	✓	n.a.
[31]	$\sim 10^{19}$	1	13	×	✓	✓	✓
[32]	$\sim 10^{25}$	1	n.a.	n.a.	n.a.	n.a.	×
[27]	$\sim 10^{25}$	1	50	✓	✓	×	n.a.
[33]	$\sim 10^5$	3	100	✓	✓	×	n.a.
[35]	variable	5	variable	✓	✓	✓	n.a.
Ours	$\sim 10^5$	7	10,000	✓	✓	✓	×

Table 1: For each reference in Section 3, we gather the conclusions of the authors on super-net correlations (GOOD COORS.), and whether WS can consistently improve NAS (WS ↗). In each entry, we specify the approximate number of architectures in the search space (SPACE SIZE), the number of search spaces considered (#SPACES), the number of architecture evaluations performed (#EVALS), whether the selection of the architecture and/or their evaluation was unbiased (UNBIASED EVALS), and whether the considered WS variant used uniform sampling (UNIFORM WS). We highlight in red what we think are shortcomings of the different analyses. "n.a." stands for not applicable, meaning that the column was not relevant to the study.

most of the NAS literature tacitly agrees on the opposite. With this study we aim at reaching a satisfactory conclusion on those two aspects in a specific setup, which is using the single-path one-shot approach of [24] on the search space described by the NAS-Bench-101 benchmark [14].

4. Methods

In this section, we describe the protocols used to address the following questions: Are WS-based proxy-accuracies significantly correlated with standalone ones? How do correlations vary with respect to the training and evaluation regimes of interest? Can the super-net find better than random

architectures? Can it find competitive architectures? How do the results vary between search spaces? Outcomes of the experiments mentioned hereafter are described in Section 5

4.1. Impact of Search Spaces on NAS Performances

To measure the impact of the search space on WS, we consider for each experiment several sub-sets of the NAS-Bench-101 search space, which we now introduce.

In NAS-Bench-101, feature maps going to the output of a cell are concatenated. Given that the shape of the output is fixed across all possible cells, and that different cells might have different number of outputs nodes, this means that the kernel depth of a cell's nodes varies over architectures, possibly hindering the use of WS. A reasonable splitting strategy is to consider the sets $(\mathcal{A}_i)_{i=1,\dots,4}$, in which architectures contain exactly i nodes connected to the output (beside the input node, which is added and not concatenated), resulting in compatible nodes with feature maps of equal sizes [1].

We also consider the full NAS-Bench-101 set, which we call $\mathcal{A}_{\text{FULL}}$: we solve the aforementioned problem by dynamically adapting the number of feature maps used for each node depending on the architecture: each node, as seen by the super-net, contains the maximum number of filters for the given layer, but sampled architectures only inherit the first n filters of the filter-bank, where n is determined so as to satisfy the constraints on the

[1] A difference of 1 feature map can still appear in $\mathcal{A}_3$ since the number of final feature maps after concatenation is rarely divisible by 3. In such case, one branch may end up with one more or one less feature map, e.g. $[42, 43, 43]$ for 128 output feature maps

output size of the architecture's cell. Other inheritance strategies could be considered, such as randomly selecting the correct number of filters from the available filter bank, but [36] notice that this scheme is both simple and efficient.

Early results additionally compelled us to study the influence of residual connections on WS. It is well known that edges connecting the input node to the output node of a cell, known as residual connections [37], significantly improve the training of convolutional architectures. Suspecting that this is also true for sub-nets of a supernet, we consider two additional search spaces: $\mathcal{A}^{\text{RES}}_{\text{FULL}}$ and $\mathcal{A}^{\text{NRES}}_{\text{FULL}}$, containing all architectures of NAS-Bench-101 with and without residual connections respectively. Figure 1 sketches the structural properties of the different search spaces.

4.2. Ranking Architectures with Weight-Sharing

With this experiment, we want to estimate the achievable correlation level between super-net based proxy accuracies, and accuracies obtained with standalone training of architectures.

We train several super-nets on the CIFAR-10 dataset. Following [24], for each mini-batch of data seen during training, a single architecture is uniformly sampled from the search space. The weights of the super-net are then updated according to the computational graph generated by the activation of this architecture.

We reuse the hyper-parameters of [14] and train the super-nets with the RMSPROP optimizer. The initial learning rate is set to 0.2 and decayed to 0 using a cosine annealing schedule over 432 training epochs, which is four times the original time budget used in NAS-Bench-101. As noted by [27]

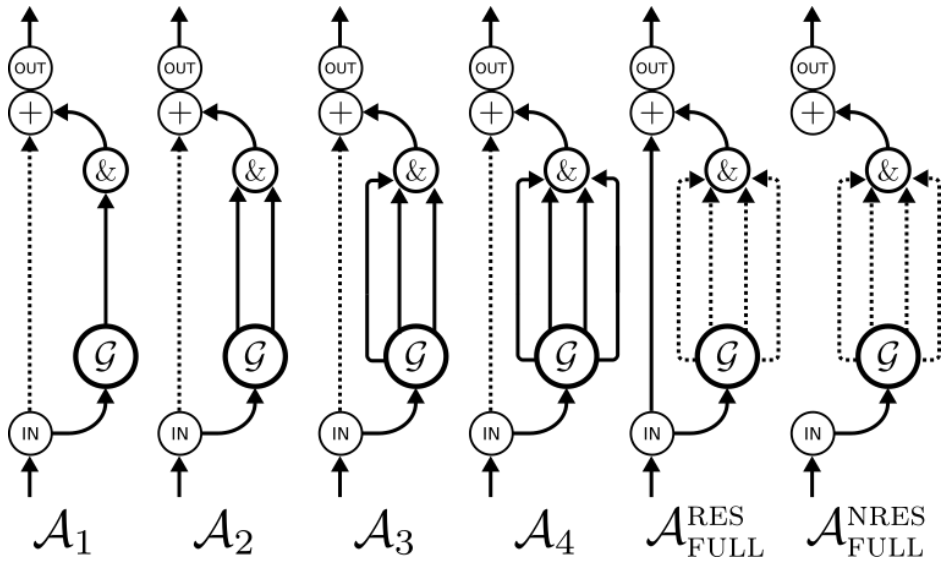


Figure 1: Structural properties of the different search-spaces. On each graph, "IN" and "OUT" denote the input and output of the cell, while "+" and "&" denote the sum and the concatenation of incoming feature maps. We only display the edges discriminating at least one search space and represent the rest of the graph with node $\mathcal{G}$. If an edge does not discriminate a specific search space, we represent it with a dotted line. $\mathcal{A}_1$ to $\mathcal{A}_4$ are characterized by the number of edges concatenated after $\mathcal{G}$, and $\mathcal{A}_{\text{FULL}}^{\text{RES}}$ and $\mathcal{A}_{\text{FULL}}^{\text{NRES}}$ by the presence or absence of a residual connection.

longer training times are often required for super-nets to converge. The momentum is set to 0.9, weight decay to 10^{-4}, and ϵ to 1. The batch size is kept to 256 and the momentum and ϵ of the batch normalization layers are respectively set to 0.003 and 10^{-5} [2]. We do not tune hyper-parameters as doing so would require the computationally expensive training of several architectures in a realistic setting.

We however reduce the initial number of filters to 16 compared to the

[2]Following PyTorch convention.

original 128, in order to accelerate training and evaluations. Earlier iterations of our experiments revealed that using the default 128 value requires fairly longer training times for the super-nets, without significantly improving the resulting correlations. Although setting it to 16 is somewhat arbitrary, [33] also note that accuracies obtained after training architectures with 16 initial filters are greatly correlated with those obtained using the baseline 128 filters. We thus consider that the number of filters is not the limiting factor of our different WS experiments. This setup additionally mimics one-shot NAS approaches such as [8, 10, 7], where the architecture found by the search is up-scaled and retrained to further improve accuracies.

We train 5 different super-nets on each search space. We then randomly sample 1,000 unique architectures from the search space and compute their proxy accuracies on the held-out validation dataset. We match those accuracies with the average validation accuracies returned by NAS-Bench-101. To quantify the quality of the correlations between the two, we use Spearman's rank correlation coefficient.

We estimate accuracies with the super-nets performing either no fine-tuning (NO-FT) or after fine-tuning the batch-norm statistics (BNS-FT). Without fine-tuning, we directly use all the parameters and batch-norm statistics of the super-net. When adapting the batch-norm statistics of the super-net to a specific architecture, we simply estimate them using a single mini-batch of training data (i.e. 256 images).

We additionally study the effect on correlations of the different training variations mentioned in Section 2.2.2. We follow the same protocol but always fine-tune the batch-norm statistics (BNS-FT). We consider four vari-

ants: averaging the gradients in Equation (9) over 3 architectures (AVG-3), sampling architectures pro-rata to their number of parameters (PRO-RATA) [27], following the single-kernel approach of [26] (SINGLE-K), and combining the single-kernel and pro-rata approaches (S-K + P-R).

4.3. Impact of Weight-Sharing on NAS Performances

Quantifying the correlation level obtained with WS on realistically sized search-spaces is of interest on its own, but is insufficient to conclude on the efficiency of WS itself. Indeed what eventually matters are not the correlations as such, but rather the quality of the architectures that can be found by exploiting them. With the experiment of this section, we aim at characterizing the interest of substituting super-net evaluations to the standalone evaluations when performing NAS.

To investigate this, we consider two very simple search algorithms: random search (RS) and a greedy local search (GS), which we both use in two different settings. In the baseline setting (BS) NAS, we directly maximize the validation accuracy of models as returned by the benchmark. In the weight-sharing (WS) setting, we maximize the performance of architectures obtained from the super-net after fine-tuning batch-norm statistics (BNS-FT).

When performing a RS, we directly sample and evaluate a fixed number of unique architectures from the given search space. The GS is a plain hill-climbing method that has been shown to be very effective on NAS benchmarks [38, 39]. A random architecture is initially sampled from the search space. All of its neighbours are evaluated, and the neighbour with the best score is selected to be the new current architecture. This process is repeated until a fixed point (an architecture whose best neighbour is itself) is reached,

after which a new random architecture is sampled. The algorithm stops when the total evaluation budget is reached. The set of neighbours of a given architecture is comprised of the architecture itself, all architectures obtained by modifying a single node operation, all valid architectures obtained by adding or removing a single edge in its adjacency matrix, as well as all valid architectures that can be obtained by adding a single node to the architecture.

In the baseline NAS setting, each search is given a budget of $10,000$ architecture evaluations. For each algorithm, we report the evolution of the test regret. The test regret is computed after each model evaluation by comparing the mean test accuracy of the model with the running best validation accuracy and the best mean test accuracy of the considered search space:

$$\mathcal{R}_{test}(t) = \max_{a \in \mathcal{A}} acc_{\text{test}}(a) - acc_{\text{test}}(\arg \max_{a \in (a_i)_0^t} acc_{\text{valid}}(a)), \tag{11}$$

where $\mathcal{A}$ is the considered search space, $a_1, \ldots, a_t$ the architectures trained on their own and evaluated on the validation dataset at time t, $acc_{\text{valid}}(a)$ and $acc_{\text{test}}(a)$ respectively refer to the validation and test accuracies of the architecture a.

When exploiting WS, the strategy is slightly different. We first optimize the architecture based on the super-net proxy accuracy. Although more evaluations could be considered, we find $10,000$ to be enough for an efficient algorithm such as GS to converge. Each evaluation requires performing an inference on a validation dataset. Although this is orders of magnitude cheaper than training architectures, we still find the process to be quite costly, requiring around 4 seconds per architecture. A search performed using the proxy

thus requires around ten hours, which, although reasonable on its own, has to be multiplied by the number of search spaces and considered seeds.

Once the optimization is finished, we assess the quality of the found solutions by querying their true validation accuracies on the benchmark. To obtain regret curves such as those of the baseline methods, we evaluate the 10,000 intermediate models considered during the WS searches in order of decreasing proxy accuracy. Again, we report the evolution of this test regret as a function of time as described in Equation (11).

The different regrets reported are plotted as a function of search time. This search time is well approximated by the duration of the training and evaluation of the different models. For a fair time-wise comparison of the regrets, we account in the WS-based paradigm for both super-nets training time and the duration of the evaluation of the 10,000 models. Unfortunately, we were not able to perform experiments using the same hardware as [14]. As a result, using the training times reported by NAS-Bench-101 would be biased, as theirs was much more efficient. To circumvent this, we used a common setup comprised of two NVidia K80 GPUs and estimated for each distinct architecture the time required to perform a single forward and a single backward pass. We averaged this quantity over three independent measures. To estimate the duration of a training, we simply multiply those quantities by the appropriate number of forward and backward passes. Besides, given that during super-net training each sampled architecture only activates the necessary parts of the network, we approximate the super-net training time by the average training time of the architectures of the considered search space.

The original purpose of WS is to perform NAS in a one-shot paradigm [40] by quickly selecting a single architecture based on proxy accuracy scores, and re-train it from scratch. Given a search space, we note for each WS run the average time spent before the first real evaluation (super-net training and evaluation, and training of the selected architecture) and report the difference between the obtained regret with the regret achieved by the baseline methods under the same time budget. To get an insight on the size of the effect, we report Cohen's d for the considered measures, which is defined as the average difference divided by the pooled standard deviation. Cohen's d [41] reflects how the measured mean absolute difference relates to the standard deviations of both populations. An effect is usually considered important if $|d|> 0.8$, mild if $|d|\approx 0.5$ and small if $|d|<0.2$ [41].

We assess the statistical significance of the differences using a Student's t-test. For the baseline searches, since there is almost no computational requirements (because a run is a simple query of the NAS-Bench-101 dataset), we follow [14] and perform 100 runs. As explained above, WS-based search takes a non-negligible amount of time. Given our computational budget, we settle on ensuring that any effect of at least medium size can be properly measured, with a statistical power of $\beta = 0.8$ and a significance level $\alpha = 0.05$. We estimate [3] that around 25 runs of the WS-based approach should grant such guarantees.

[3] using the STATSMODELS python package [42]

4.4. Good Practices

As pointed out in Section 3 and Table 1, existing analyses of WS in the literature bear several limits. In this section we present what we think are good practices for evaluating the impact of WS on NAS. With those guidelines, we hope to pave the way for future research and possible analyses of other benchmarks. We further succinctly indicate in Table 2 the strategy that we employed to avoid each pitfall.

Search Space Variability: When evaluating WS, we suggest not relying on a single search space, but rather explore various search spaces. This diversity gives an idea of the intrinsic variance associated with the procedure. As discussed in Section 5.3 this multiplicity is key as WS performance varies greatly with the explored search space.

Unbiased Estimation of Correlations: To fairly assess the quality of correlations between super-net and standalone scores, one must limit biases that could emerge from the super-net in three respects: sampling biases during training that result in some architectures being trained more often than others, biases in the selection of architectures during evaluation that result in correlations not representative of the whole search space, and biases in the evaluation procedure itself that result in poor non-representative individual performance.

Visualization of Correlations: Correlation metrics such as Spearman's rank correlation coefficient cannot capture the complexity of a plain scatter

plot of predicted against true score. Such scatter plots additionally reveal over or under evaluation biases created by the super-net. Examples of such figures can be observed in Figure 3 and Figure 4.

Assessment of Computation Times: When reporting the evolution of regrets as a function of computing time, all sources of computations must be included. This includes the time required to train the super-net, as well as to perform the individual evaluation of all the architectures of interest.

Satisfying Statistical Power: The proper statistical comparison of two NAS approaches is crucial to reach any conclusion. Unfortunately, it is quite common in the NAS literature to claim the superiority of NAS method based on a single run. One must make sure that enough runs are considered to get statistically significant comparisons [43].

5. Results and Discussion

We now describe the results of the above studies. We then discuss the influence of the search space on WS.

5.1. Ranking Capabilities of Weight-Sharing

For each search space, we report in the left part of Table 3 the average over 5 different super-nets of the rank correlation between the standalone accuracy returned by NAS-Bench-101 and the proxy accuracies obtained after applying the two evaluation protocols described in Section 4.2. NO-FT refers to performing no fine-tuning, and BNS-FT to fine-tuning the batch-norm statistics. Without any fine-tuning, correlations are poor across all

Good Practice	Our Approach	Observed Results
Search Space Variability	We explored seven different sub search-spaces of NAS-Bench-101.	Correlations and WS performance vary greatly with the considered search space. For instance, the Spearman coefficient can range between 0.46 and 0.71.
Unbiased Estimation of Correlations	We used the uniform sampling scheme of [24].	Uniformly sampling architectures during the training is a baseline that allowed us to measure the contribution of other more advanced techniques.
	We uniformly sampled architectures from the considered search space.	Uniformly sampling architectures during evaluation allowed to conclude of the correlation capabilities of WS, as well as identifying some biases in the evaluation of certain architectures
	We tuned batch-norm statistics during evaluation.	The average correlation over the different search spaces increases by 270% when fine-tuning batch-norm statistics.
Visualization of Correlations	We introduced scatter plots with colors indicating specific properties.	We identified some biases in the evaluation of certain architectures.
Assessment of Computation Times	We accounted for super-net training and evaluation times.	Training super-nets takes a non-negligible amount of time and accounting for this duration makes a baseline random search equivalent to a random search employing WS on search spaces.
Satisfying Statistical Power	We met our statistical power and significance goals with 25 runs.	This allowed us to conclude on the efficiency of WS against a random search baseline.

Table 2: For each good practice introduced in Section 4.4, we indicate the strategy that we employed and the observed results.

	NO FT	BNS-FT	SINGLE-K	PRO-RATA	AVG-3	S-K + P-R
$\mathcal{A}_4$	0.08 ± 0.17	**0.64 ± 0.03**	0.66 ± 0.01	0.68 ± 0.03	∗ 0.67 ± 0.02	**0.69 ± 0.02**
$\mathcal{A}_3$	0.12 ± 0.15	**0.59 ± 0.03**	0.62 ± 0.02	0.63 ± 0.02	0.61 ± 0.03	**0.66 ± 0.02**
$\mathcal{A}_2$	0.24 ± 0.03	**0.60 ± 0.04**	0.64 ± 0.02	0.64 ± 0.02	0.61 ± 0.01	**0.65 ± 0.02**
$\mathcal{A}_1$	0.32 ± 0.05	**0.68 ± 0.02**	0.72 ± 0.02	**0.75 ± 0.02**	0.73 ± 0.01	0.67 ± 0.01
$\mathcal{A}_{\text{FULL}}$	0.24 ± 0.05	**0.56 ± 0.04**	∗ **0.63 ± 0.02**	0.59 ± 0.03	0.61 ± 0.02	0.58 ± 0.02
$\mathcal{A}_{\text{FULL}}^{\text{NRES}}$	∗ 0.11 ± 0.05	∗ **0.46 ± 0.06**	∗ **0.58 ± 0.02**	0.56 ± 0.03	0.52 ± 0.02	0.49 ± 0.02
$\mathcal{A}_{\text{FULL}}^{\text{RES}}$	0.34 ± 0.10	**0.71 ± 0.02**	0.68 ± 0.01	**0.72 ± 0.02**	0.69 ± 0.02	0.66 ± 0.01

Table 3: Spearman's Rank Correlation Coefficient Between WS and Standalone Evaluations for Various Search Spaces, WS Variants, and Evaluation Schemes . We report the average over 5 independent runs and the 95% confidence interval for its estimation. On the left, we use the baseline WS approach and the perform either no fine-tuning, or batch-norm statistics fine-tuning. On the right, we test some variants of WS described in Section 2.2.2 and always fine-tune batch-norm statistics during evaluations. Results marked with an asterisk ∗ indicate that one of the super-net failed to converge, and that the reported statistics are computed using only the four others.

search-spaces, with substantial variances. With batch-norm statistics fine-tuning, the average correlation increases by 270% over the NO-FT scheme, granting almost 3 times better results on average.

In the right part of Table 3, we present the average rank correlations obtained from training the super-net with the different variants described in Section 2.2.2, and applying batch-norm statistics fine-tuning during evaluations. SINGLE-K refers to exploiting the single kernel variant of [26], PRO-RATA to sampling architectures pro-rata to their number of parameters [27], and S-K + P-R the combination of the two. AVG-3 refers to averaging gradi-

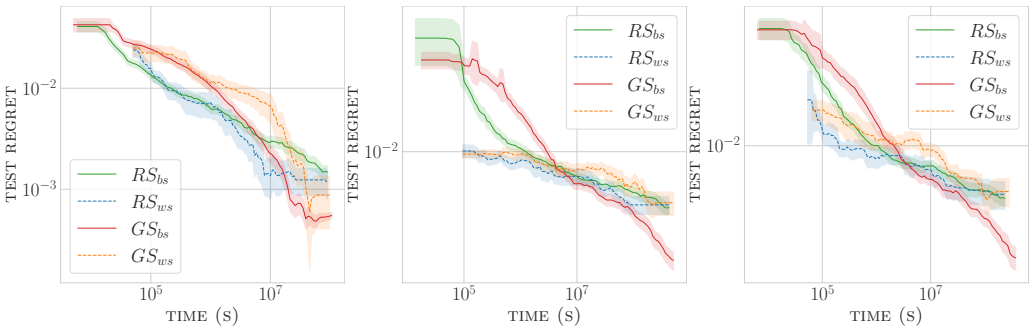
ents during the training of the super-net over three architectures. We notice that all approaches lead to a small improvement of the correlations, as well as a slight variance reduction.

These simple results show that it is possible to get correlations between proxy evaluations performed with WS and full-budget evaluations as long as batch-norm statistics are adapted to the evaluated architectures. We notice that all the works mentioning poor correlations in Section 3 do not detail their evaluation setup, and we suspect that they do not adapt batch-norm statistics. Additionally, it is possible to further improve the resulting correlations by modifying the super-net training in various ways.

5.2. Can Weight-Sharing Improve NAS ?

The regret curves of the different NAS experiments described in Section 4.3 are reported in Figure 2. The relative position of the different methods largely depends on the considered search space, but one can see that given sufficient training time, baseline GS outperforms all other algorithms. In a one-shot setting, (i.e. given a time budget equivalent to the evaluation of a single WS-picked architecture), WS-based approaches seem to perform better on average than the baseline RS and GS but can be quite unreliable.

We report in Table 4 the average regret difference between the baseline random search and the weight-sharing based random search ($RS_{bs} - RS_{ws}$), and between the baseline random search and the weight-sharing based greedy search ($RS_{bs} - GS_{ws}$), in the one-shot setting. Numerical and statistical results coincide with the visual results of Figure 2. Similarly, we report in Table 5 the average regret difference between the baseline greedy search and the weight-sharing based random search ($GS_{bs} - RS_{ws}$), and between the



(a) $\mathcal{A}_4$ (b) $\mathcal{A}_1$ (c) $\mathcal{A}_{\text{FULL}}$

Figure 2: We report for a few search space the test regret as a function of time for the different NAS algorithms considered. Curves are averaged over 100 runs for RS_{bs} and GS_{bs}, and 25 runs for RS_{ws} and GS_{ws}. Visible colored areas correspond to the 95% confidence interval for the estimation of the average. Notice that both axes use a logarithmic scale. Figures for all search spaces can be found in Appendix B.2.

baseline greedy search and the weight-sharing based greedy search ($GS_{bs} - GS_{ws}$).

Combining WS and random search results in a significant improvement over the baseline random search (left column of Table 4) on $\mathcal{A}_1$ ($d = +1.09, p = 0.00$), $\mathcal{A}^{\text{NRES}}_{\text{FULL}}$ ($d = +0.48, p = 0.03$) and $\mathcal{A}^{\text{RES}}_{\text{FULL}}$ ($d = +0.63, p = 0.01$), but grants significant worse results on $\mathcal{A}_4$ ($d = -0.85, p = 0.00$). Results on $\mathcal{A}_3$ ($d = -0.26, p = 0.25$), $\mathcal{A}_2$ ($d = +0.42, p = 0.07$) and $\mathcal{A}_{\text{FULL}}$ ($+0.44, p = 0.06$), are below our effect size threshold and non-significant. Unsurprisingly, the baseline greedy search performs worse than the baseline random search in the one-shot setting, as can be deduced from the left column of Table 5, where GS is outperformed on all but one search space. This is to be expected from an algorithm without any direct exploration of the search space. Results in Table 4 moreover suggest that combining WS with GS rather than RS re-

	$RS_{bs} - RS_{ws}$	$RS_{bs} - GS_{ws}$
$\mathcal{A}_4$	-0.68 ± 0.80 (p=0.00, d=-0.85)	-0.82 ± 0.83 (p=0.00, d=-0.99)
$\mathcal{A}_3$	-0.20 ± 0.77 (p=0.25, d=-0.26)	-0.62 ± 0.88 (p=0.00, d=-0.71)
$\mathcal{A}_2$	0.36 ± 0.87 (p=0.07, d=$+0.42$)	0.54 ± 0.85 (p=0.01, d=$+0.63$)
$\mathcal{A}_1$	1.18 ± 1.08 (p=0.00, d=$+1.09$)	1.30 ± 1.10 (p=0.00, d=$+1.18$)
$\mathcal{A}_{\text{FULL}}$	0.53 ± 1.22 (p=0.06, d=$+0.44$)	0.76 ± 0.96 (p=0.00, d=$+0.79$)
$\mathcal{A}_{\text{FULL}}^{\text{NRES}}$	0.63 ± 1.30 (p=0.03, d=$+0.48$)	0.32 ± 1.44 (p=0.32, d=$+0.22$)
$\mathcal{A}_{\text{FULL}}^{\text{RES}}$	0.33 ± 0.52 (p=0.01, d=$+0.63$)	-0.02 ± 0.58 (p=0.87, d=-0.04)

Table 4: We report the average regret differences between the baseline random search and the WS-based random search ($RS_{bs} - RS_{ws}$), and between the baseline random search and the WS-based greedy search ($RS_{bs} - GS_{ws}$) in the one-shot paradigm. We additionally report the pooled standard deviation, the p-value, as well as the effect size d. **For clarity purposes, regrets are multiplied by 100**. We test for the statistical significance of the difference using an independent t-test and report the resulting p-values. Results highlighted in blue correspond to settings where the considered method performed significantly worse than the random search baseline ($p < 0.05$), whereas results marked in red highlight settings in which the considered method performed significantly better than random search baseline ($p < 0.05$).

sults in unexpected behaviors, as it can either improve ($\mathcal{A}_2$, $\mathcal{A}_1$, $\mathcal{A}_{\text{FULL}}$), or be detrimental to ($\mathcal{A}_4$, $\mathcal{A}_3$, $\mathcal{A}_{\text{FULL}}^{\text{NRES}}$, $\mathcal{A}_{\text{FULL}}^{\text{RES}}$) performances depending on the search space.

All in all, results suggest that in a one-shot setting, WS can improve the performance of RS but that its efficiency is inconsistent and on average relatively small. To put the different reported regrets in context, one can consider that with an effect size $d = +0.44$ for the WS based RS over baseline

	$GS_{bs} - RS_{ws}$	$GS_{bs} - GS_{ws}$
$\mathcal{A}_4$	0.30 ± 1.21 (p=0.27, d=+0.25)	0.19 ± 1.25 (p=0.51, d=+0.15)
$\mathcal{A}_3$	0.72 ± 1.45 (p=0.03, d=+0.50)	0.28 ± 1.50 (p=0.41, d=+0.18)
$\mathcal{A}_2$	1.67 ± 2.92 (p=0.01, d=+0.57)	1.86 ± 2.92 (p=0.01, d=+0.64)
$\mathcal{A}_1$	2.46 ± 1.85 (p=0.00, d=+1.33)	2.50 ± 1.84 (p=0.00, d=+1.36)
$\mathcal{A}_{\text{FULL}}$	1.42 ± 3.03 (p=0.04, d=+0.47)	1.67 ± 2.95 (p=0.01, d=+0.57)
$\mathcal{A}_{\text{FULL}}^{\text{NRES}}$	1.48 ± 1.76 (p=0.00, d=+0.84)	1.14 ± 1.85 (p=0.01, d=+0.62)
$\mathcal{A}_{\text{FULL}}^{\text{RES}}$	0.78 ± 0.87 (p=0.00, d=+0.90)	0.43 ± 0.93 (p=0.04, d=+0.47)

Table 5: We report the average regret differences between the baseline greedy search and the WS-based random search ($GS_{bs} - RS_{ws}$), and between the baseline greedy search and the WS-based greedy search ($GS_{bs} - GS_{ws}$) in the one-shot paradigm. We additionally report the pooled standard deviation, the p-value, as well as the effect size d. **For clarity purposes, regrets are multiplied by 100**. We test for the statistical significance of the difference using an independent t-test and report the resulting p-values. Results highlighted in blue correspond to settings where the considered method performed significantly worse than the greedy search baseline ($p < 0.05$), whereas results marked in red highlight settings in which the considered method performed significantly better than the greedy search baseline ($p < 0.05$).

RS on $\mathcal{A}_{\text{FULL}}$, the probability that a random run with WS produces a smaller regret than the baseline given the same time budget is only around 61%[4]. On $\mathcal{A}_1$, where WS is somehow very effective and produces a large effect size of $d = +1.09$, this probability reaches a maximum of 78%. On the contrary,

[4]An interactive visualization of the phenomenon can be studied at `https://rpsychologist.com/d3/cohend/` [44]

on $\mathcal{A}_4$, where WS is least effective, this probability can get as low as 7%. As it has been noted several times in the literature [45, 32, 28], reporting the results over several runs is thus crucial to NAS research, especially when considering moderate or small effect sizes.

Interestingly, Table 4 reveals no clear link between the average level of correlation reached by WS on a search space and its ability to outperform a baseline RS in a one-shot setting: on $\mathcal{A}_{\text{FULL}}^{\text{NRES}}$, where correlations in Table 3 are the lowest, WS significantly outperforms RS, whereas it offers terrible results on $\mathcal{A}_4$ despite significantly better correlations. WS offers similar correlations on $\mathcal{A}_2$ and $\mathcal{A}_3$, but the WS-guided greedy search respectively gives smaller and larger regrets than the baseline random search. On $\mathcal{A}_{\text{FULL}}^{\text{RES}}$ and $\mathcal{A}_1$, where WS offers the best correlations, the WS-guided greedy search is respectively equivalent to RS, and much better than RS.

Under the time constraints of one-shot NAS, WS can slightly outperform a baseline RS, although rarely to a significant extent, and can even be worse. Besides, there seems to be no obvious relationship between the level of correlation between proxy and standalone evaluations, and the performances of WS on a search space.

5.3. Variations between Search Spaces

From Section 5.2, WS-guided NAS seems to often slightly outperform RS, but this depends on the search space.

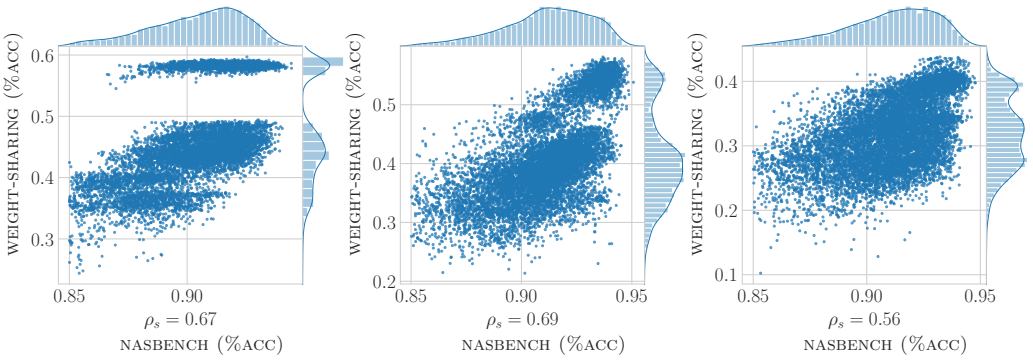
Coincidentally, we notice from the results of Section 5.1 that simply changing the number of nodes connected to the output makes the average correlation vary between 0.59 on $\mathcal{A}_3$ and 0.68 on $\mathcal{A}_1$. Additionally, restricting the search space to architectures presenting a residual connection has a

noticeable positive effect on the correlations, as they increase from 0.46 to 0.71 between $\mathcal{A}_{\text{FULL}}^{\text{NRES}}$ and $\mathcal{A}_{\text{FULL}}^{\text{RES}}$. The search space itself has an important impact on the correlations, even more so when using the training enhancements described in Section 2.2.2.

The size of the datasets could explain the varying correlations. It has often been asserted in the literature that, the more architectures there are in the search space, the harder it is to train the super-net. The Spearman rank's correlation between the average correlation obtained with batch-size fine-tuning (BNS-FT) reported in Table 3 and the sizes of the dataset reaches -0.71 ($p = 0.07$). The effect hints that larger search-spaces could possibly lead to smaller correlations between proxy and standalone evaluations, but the relatively low number of search spaces of this study prevents us from positively rejecting the null hypothesis that it does not with great confidence, and further studies are required to conclude on this matter. Besides, results in Section 5.1 suggest that it is probably not the only aspect of the search space that is of influence. On $\mathcal{A}_2$ and $\mathcal{A}_3$, WS offers roughly the same level of correlation, despite $\mathcal{A}_2$ being twice larger than $\mathcal{A}_3$. The correlation achieved is 25% smaller in $\mathcal{A}_{\text{FULL}}^{\text{NRES}}$ than in $\mathcal{A}_{\text{FULL}}$, with 23% less architectures. It is also interesting to note that few architectures are actually seen during training: given 432 training epochs of 157 mini-batches of data, less than $67,824$ unique architectures are used to update the super-net. This might be enough to cover $\mathcal{A}_4$ or $\mathcal{A}_{\text{FULL}}^{\text{RES}}$, but represents only a tiny fraction of larger datasets, such as $\mathcal{A}_2$ ($\simeq 200,000$ architectures), or $\mathcal{A}_{\text{FULL}}$ ($\simeq 400,000$ architectures). Further studies are required to clearly establish whether the size of the dataset has a non-negligible impact on the correlation capabilities of WS, but several facts

suggest that it cannot entirely explain the discrepancies between the different search spaces.

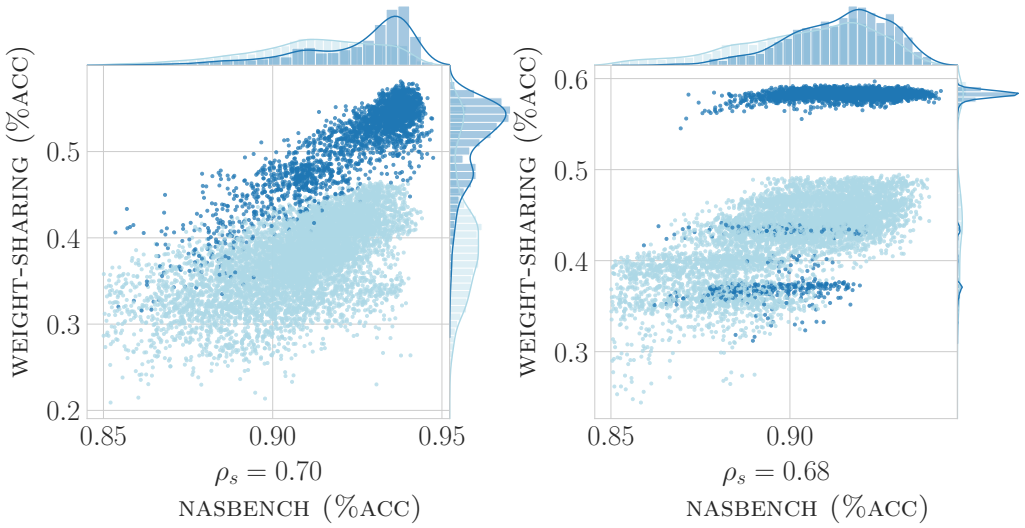
We report in Figure 3 and for each search space a scatter plot between the true validation accuracies and the proxy accuracies resulting from training a super-net. Interestingly, several visible clusters seem to be linked to proxy evaluations. For each scatter-plot, we report the distributions of the true validation and proxy accuracies over sampled architectures. Coincidentally with the different visible architecture clusters, distributions of proxy evaluations are much less regular than their true validation counterparts, often presenting several modes. The clusters of architectures in the scatter-plots visually transcribe existing biases in proxy evaluations.



(a) $\mathcal{A}_4$ (b) $\mathcal{A}_1$ (c) $\mathcal{A}_{\text{FULL}}$

Figure 3: We report, for a few search spaces, a scatter plot of the proxy accuracy computed using a super-net BNS-FT (y-axis), and the average validation accuracy returned by NAS-Bench-101(x-axis) for 10,000 architectures. Scatter plots for all search spaces can be found in Appendix B.1.

There is no trivial relation between different biases and particular structural properties of the architectures. Fortunately, some biases are easier to



(a) $\mathcal{A}_1$ (b) $\mathcal{A}_4$

Figure 4: We report for a super-net trained on $\mathcal{A}_1$ (left) and $\mathcal{A}_4$ (right) the proxy accuracy computed after fine-tuning the batch-norm statistics (y-axis), and the average validation accuracy returned by NAS-Bench-101 (x-axis) for $1,000$ architectures. We highlight in a darker tone the points corresponding to architectures with residual connections (left) and architectures with a 3×3 convolution on the first node (right). Both examples reveal a clear bias in super-net evaluations. We also report the distributions of the proxy and standalone accuracies of the sampled architectures for the complete population and for the sub-population of interest.

highlight than others. We focus on two such biases in Figure 4. On $\mathcal{A}_1$, architectures with a residual connection tend to get better evaluations than those without. On $\mathcal{A}_4$, the presence of a 3×3 convolution on the first node triggers over-evaluation. Such clusters can be seen in the scatter plots of all search spaces except $\mathcal{A}^{\text{RES}}_{\text{FULL}}$. Different search spaces bias the super-nets in different ways, resulting in different structural patterns of over/under-evaluations.

The patterns appearing in the scatter-plots may explain the search results

of Section 5.2 better than the correlations level reached by WS. On $\mathcal{A}_4$, the over-evaluation bias visible in Figure 4 creates a cluster of architectures with excellent proxy accuracies. As a result, in a one-shot paradigm, WS neglects a large number of architectures with equal or better capabilities that random search does not miss. Although the cluster contains a few of the best architectures, its average standalone accuracy is particularly poor. This impedes WS from selecting top models, and makes the early WS-guided search worse than RS on this particular search space. On $\mathcal{A}_1$, the over-evaluation bias towards residual connections benefits to the search, as architectures with residual connections are better on average and constitute most of the best architectures of the search space. The WS-guided search is in turn quite efficient.

The patterns of over/under-evaluations dictate the search behavior when exploiting WS. If WS is biased towards interesting patterns in the considered search space, then it is likely to perform much better than random search. Otherwise, the difference may not be significant. In the worst scenario, the bias can even be strong enough to undermine the performances of WS.

6. Conclusion

In this paper we have leveraged the NAS-Bench-101 dataset to investigate the impact of WS on the efficiency of NAS over seven search spaces. Our results lead to the following conclusions. First, super-nets trained with WS can offer significant correlations between proxy evaluations and standalone evaluations, but fine-tuning the batch-norm statistics of the models is mandatory for the process to be successful. The results can be further

improved by tweaking the WS training process, but the search space itself has a more significant influence over the quality of correlations.

Most importantly, in a one-shot NAS setting, WS combined with RS is not consistently faster than a baseline RS. Improvements are, to a large extent, search-space dependent and of relatively small effect. Super-nets resulting from optimization with WS can be biased towards specific structural patterns in the architectures, which also vary depending on the search-space. Those patterns, rather than the level of correlations, seem to dictate the efficiency of NAS when exploiting WS. Given that each search space has its own specific biases, it is hard to foresee how well WS is going to perform. This conclusion goes against the tacit agreement in the literature that WE based NAS algorithms are both efficient and effective.

Gathering both theoretical understanding and additional experimental evidence of the different biases created by search spaces on the super-net, such as done in [46] will be key to the weight-sharing literature and the successful application of NAS, and is a promising lead for future work. Additionally, It would be interesting to establish whether the described results would transfer from CIFAR-10 to larger image datasets such as IMAGENET, or to larger search spaces. Both perspective constitute excellent material for further studies, and exploiting different NAS benchmarks, such as NAS-Bench-201 [47] and NAS-Bench-301 [48], would likely be key to future work on the subject. To facilitate future research on those matters, we have introduced in this paper several good practices which we identified as key to a proper analysis of the performance of WS. We trust that with those good practices in mind, future researchers will avoid pitfalls that would limit the scope of their results.

7. Acknowledgements

We thank Arnaud Dapogny for feedback on an earlier version of this work. Financial support for this study was provided by Gleamer, which had no involvement in the design of the study, nor on the collection, analysis and interpretation of the data.

8. Vitae

Aloïs Pourchot received the M.S.E. degree from Télécom Paris, Paris, France, in 2019 and a M.S. in Computer Vision and Artificial Intelligence from ENS Paris Saclay, Paris, France in 2019. He is currently pursuing the Ph.D. degree with Sorbonne University, Paris, France, under the supervision of Prof. O. Sigaud. As part of the CIFRE program he is also an employee of Gleamer, Paris, France. His current research interests are computer vision, neural architecture search, and medical image processing.

Kevin Bailly is associate professor with the Institute of Intelligent Systems and Robotics (ISIR) at Sorbonne University and Head of Research of Datakalab. He received the PhD degree in computer science from the Pierre et Marie Curie University in 2010 and was a postdoctoral

researcher at Telecom Paris from 2010 to 2011. His research interests are in machine learning and computer vision applied to face processing, behavior analysis and medical image processing.


Alexis Ducarouge received the M.S.E. degree from Télécom Paris, France in 2017 and a M.S. in Cognitive Sciences and Artificial Intelligence from ENS Ulm, Paris, France in 2017. He worked on Deep Reinforcement Learning approaches during his internship at the robotics institute of Sorbonne university (ISIR) under the supervision of Prof. O. Sigaud. He co-founded Gleamer, in 2017 where he holds the position of Scientific Technical Director.


Olivier Sigaud was born in 1968 in France. He received a Ph.D. degree in computer science with Paris XI University, Orsay, France in 1996 and a Ph.D. degree in philosophy with Paris I University, Paris, France in 2002. From 1996 to 2001, he was with Dassault Aviation, S^t-Cloud, France. He is currently Professor in computer science at Sorbonne University and member of the robotics institute of this university (ISIR). He has published numerous articles and has been involved in several national

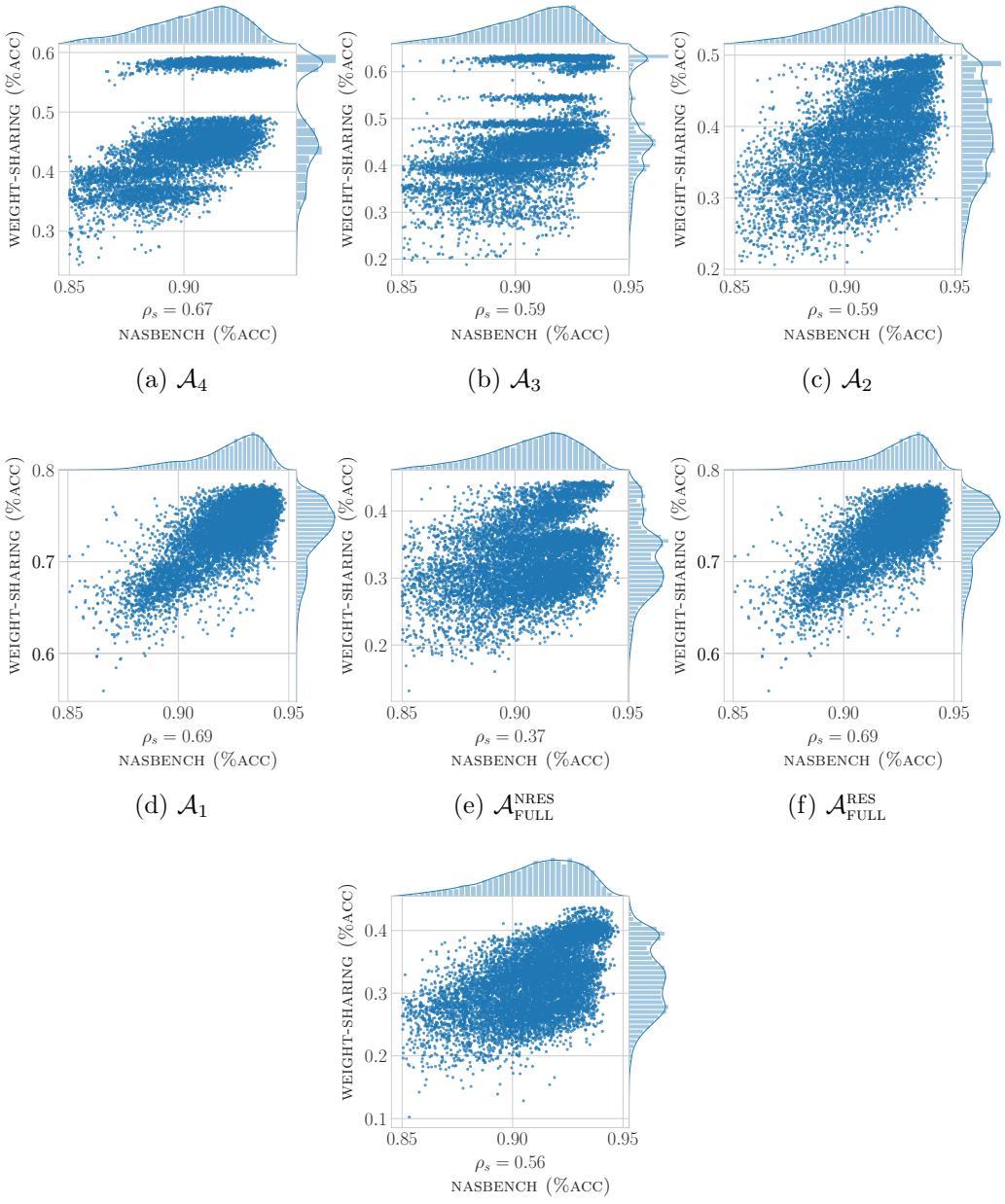
and European projects.

Appendix A. Additional Scatter Plots

We provide scatter plots of the proxy accuracy computed using a supernet and the average validation accuracy return by the NAS-Bench-101 benchmark for all search spaces in Figure B.1.

Appendix B. Additional Regret Plots

We provide the evolution of the test regret as a function of time for the different NAS algorithms considered in 4.3 in Figure B.2.

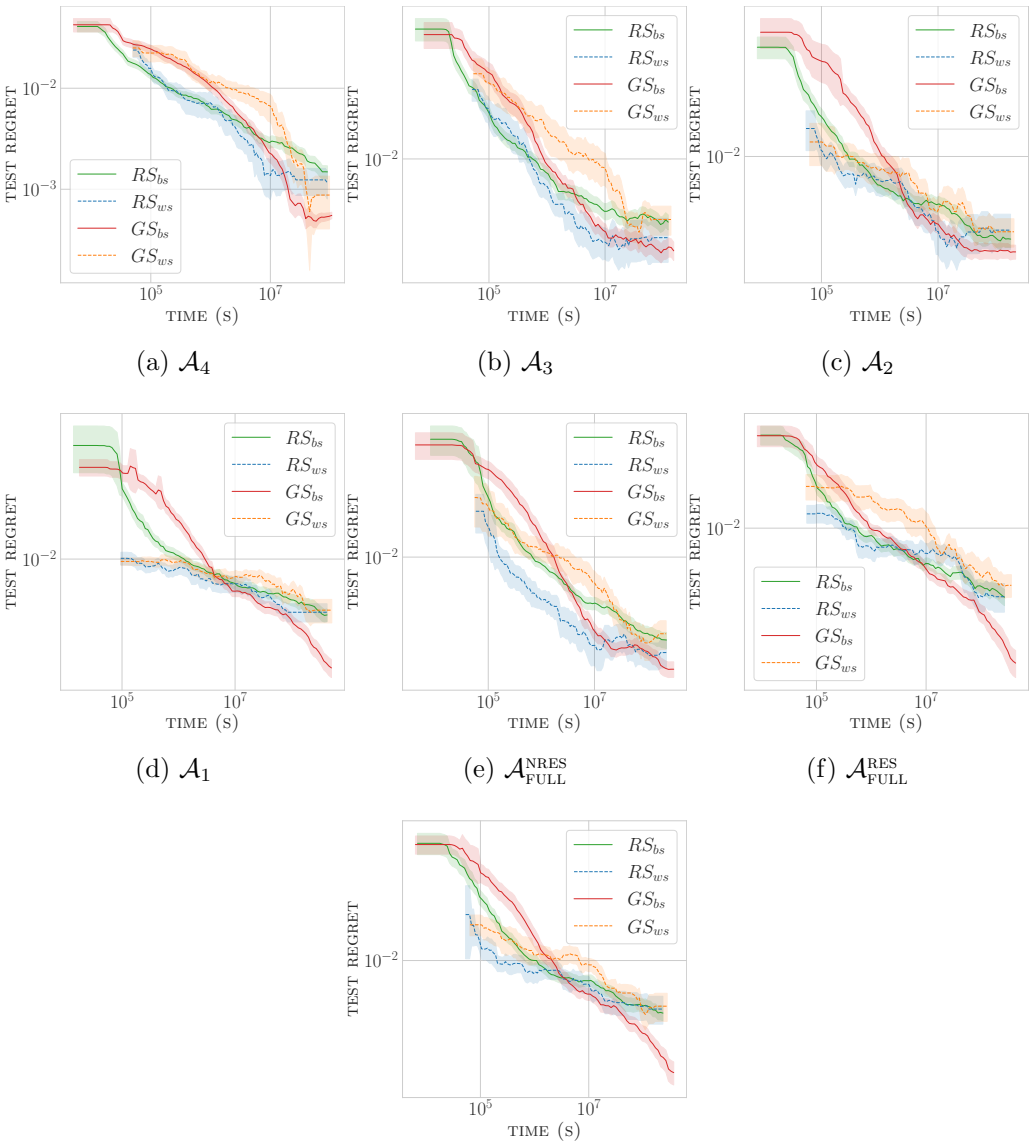


(a) $\mathcal{A}_4$ (b) $\mathcal{A}_3$ (c) $\mathcal{A}_2$

(d) $\mathcal{A}_1$ (e) $\mathcal{A}^{\text{NRES}}_{\text{FULL}}$ (f) $\mathcal{A}^{\text{RES}}_{\text{FULL}}$

(g) $\mathcal{A}_{\text{FULL}}$

Figure B.1: For each search space we report a scatter plot of the proxy accuracy computed using a super-net BNS-FT (y-axis), and the average validation accuracy returned by NAS-Bench-101(x-axis) for 10,000 architectures.



(a) $\mathcal{A}_4$ (b) $\mathcal{A}_3$ (c) $\mathcal{A}_2$

(d) $\mathcal{A}_1$ (e) $\mathcal{A}^{\text{NRES}}_{\text{FULL}}$ (f) $\mathcal{A}^{\text{RES}}_{\text{FULL}}$

(g) $\mathcal{A}_{\text{FULL}}$

Figure B.2: For each search space, we report the evolution of the test regret as a function of time for the different NAS algorithms considered. Curves are averaged over 100 runs for RS_{bs} and GS_{bs}, and 25 runs for RS_{ws} and GS_{ws}. Visible colored areas correspond to the 95% confidence interval for the estimation of the average. Notice that both axes use a logarithmic scale..

References

[1] R. Salakhutdinov, Deep learning, in: S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, R. Ghani (Eds.), The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014, ACM, 2014, p. 1973. doi:10.1145/2623330.2630809.
URL https://doi.org/10.1145/2623330.2630809

[2] T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: A survey, The Journal of Machine Learning Research 20 (1) (2019) 1997–2017.

[3] M. Wistuba, A. Rawat, T. Pedapati, A survey on neural architecture search, arXiv preprint arXiv:1905.01392 (2019).

[4] E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press, 2019, pp. 4780–4789. doi:10.1609/aaai.v33i01.33014780.
URL https://doi.org/10.1609/aaai.v33i01.33014780

[5] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake

City, UT, USA, June 18-22, 2018, IEEE Computer Society, 2018, pp. 8697–8710. doi:10.1109/CVPR.2018.00907.
URL http://openaccess.thecvf.com/content_cvpr_2018/html/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.html

[6] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.
URL https://openreview.net/forum?id=r1Ue8Hcxg

[7] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, J. Dean, Efficient neural architecture search via parameter sharing, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 4092–4101.
URL http://proceedings.mlr.press/v80/pham18a.html

[8] H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.
URL https://openreview.net/forum?id=S1eYHoC5FX

[9] S. Xie, H. Zheng, C. Liu, L. Lin, SNAS: stochastic neural architecture search, in: 7th International Conference on Learning Representations,

ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.
URL https://openreview.net/forum?id=rylqooRqK7

[10] F. P. Casale, J. Gordon, N. Fusi, Probabilistic Neural Architecture Search, arXiv e-prints (2019) arXiv:1902.05116arXiv:1902.05116.

[11] A. Brock, T. Lim, J. M. Ritchie, N. Weston, SMASH: one-shot model architecture search through hypernetworks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018.
URL https://openreview.net/forum?id=rydeCEhs-

[12] S. J. Nowlan, G. E. Hinton, Simplifying neural networks by soft weight-sharing, Neural computation 4 (4) (1992) 473–493.

[13] Z. Tang, R. Zhu, P. Lin, J. He, H. Wang, Q. Huang, S. Chang, Q. Ma, A hardware friendly unsupervised memristive neural network with weight sharing mechanism, Neurocomputing 332 (2019) 193–202.

[14] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, F. Hutter, Nas-bench-101: Towards reproducible neural architecture search, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, Vol. 97 of Proceedings of Machine Learning Research, PMLR, 2019, pp. 7105–7114.
URL http://proceedings.mlr.press/v97/ying19a.html

[15] B. Colson, P. Marcotte, G. Savard, An overview of bilevel optimization, Annals of operations research 153 (1) (2007) 235–256.

[16] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, M. Pontil, Bilevel programming for hyperparameter optimization and meta-learning, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1563–1572.
URL http://proceedings.mlr.press/v80/franceschi18a.html

[17] T. Elsken, J. H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.
URL https://openreview.net/forum?id=ByME42AqK7

[18] L. Wang, Y. Zhao, Y. Jinnai, Alphax: exploring neural architectures with deep neural networks and monte carlo tree search, CoRR abs/1805.07440 (2018). arXiv:1805.07440.
URL http://arxiv.org/abs/1805.07440

[19] S. Falkner, A. Klein, F. Hutter, BOHB: robust and efficient hyperparameter optimization at scale, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1436–

1445.
URL http://proceedings.mlr.press/v80/falkner18a.html

[20] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, Z. Li, DARTS+: Improved Differentiable Architecture Search with Early Stopping, arXiv e-prints (2019) arXiv:1909.06035arXiv:1909.06035.

[21] Y. Xu, L. Xie, X. Zhang, X. Chen, G. Qi, Q. Tian, H. Xiong, PC-DARTS: partial channel connections for memory-efficient differentiable architecture search, CoRR abs/1907.05737 (2019). arXiv:1907.05737.
URL http://arxiv.org/abs/1907.05737

[22] Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, K. Nishida, Adaptive stochastic natural gradient method for one-shot neural architecture search, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, Vol. 97 of Proceedings of Machine Learning Research, PMLR, 2019, pp. 171–180.
URL http://proceedings.mlr.press/v97/akimoto19a.html

[23] H.-G. Beyer, H.-P. Schwefel, Evolution strategies–a comprehensive introduction, Natural computing 1 (1) (2002) 3–52.

[24] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, J. Sun, Single path one-shot neural architecture search with uniform sampling, in: European Conference on Computer Vision, Springer, 2020, pp. 544–560.

[25] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, Q. V. Le, Understanding and simplifying one-shot architecture search, in: J. G. Dy,

A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 549–558.
URL http://proceedings.mlr.press/v80/bender18a.html

[26] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, D. Marculescu, Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours, arXiv e-prints (2019) arXiv:1904.02877arXiv:1904.02877.

[27] R. Luo, T. Qin, E. Chen, Balanced one-shot neural architecture optimization, arXiv preprint arXiv:1909.10815 (2019).

[28] K. Yu, C. Sciuto, M. Jaggi, C. Musat, M. Salzmann, Evaluating the search phase of neural architecture search, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.
URL https://openreview.net/forum?id=H1loF2NFwr

[29] R. Luo, F. Tian, T. Qin, E. Chen, T. Liu, Neural architecture optimization, in: S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 7827–7838.
URL https://proceedings.neurips.cc/paper/2018/hash/933670f1ac8ba969f32989c312faba75-Abstract.html

[30] Y. Zhang, Q. Zhang, J. Jiang, Z. Lin, Y. Wang, Deeper insights into weight sharing in neural architecture search, in: Submitted to International Conference on Learning Representations, 2020, rejected.
URL https://openreview.net/forum?id=ryxmrpNtvH

[31] X. Chu, B. Zhang, R. Xu, J. Li, FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search, arXiv e-prints (2019) arXiv:1907.01845arXiv:1907.01845.

[32] A. Yang, P. M. Esperança, F. M. Carlucci, NAS evaluation is frustratingly hard, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.
URL https://openreview.net/forum?id=HygrdpVKvr

[33] A. Zela, J. Siems, F. Hutter, Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.
URL https://openreview.net/forum?id=SJx9ngStPH

[34] A. Pourchot, A. Ducarouge, O. Sigaud, To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing, arXiv e-prints (2020) arXiv:2002.04289arXiv:2002.04289.

[35] Y. Zhang, Q. Zhang, Y. Yang, How Does Supernet Help in Neural Architecture Search?, arXiv e-prints (2020) arXiv:2010.08219arXiv:2010.08219.

[36] K. Yu, R. Ranftl, M. Salzmann, How to train your super-net: An analysis of training heuristics in weight-sharing nas, arXiv preprint arXiv:2003.04276 (2020).

[37] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 770–778. doi:10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90

[38] C. White, S. Nolen, Y. Savani, Local search is state of the art for nas benchmarks, arXiv preprint arXiv:2005.02960 (2020).

[39] T. D. Ottelander, A. Dushatskiy, M. Virgolin, P. A. Bosman, Local search is a remarkably strong baseline for neural architecture search, arXiv preprint arXiv:2004.08996 (2020).

[40] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, Z. Chen, L. Wang, A. Xiao, J. Chang, X. Zhang, et al., Weight-sharing neural architecture search:a battle to shrink the optimization gap, arXiv preprint arXiv:2008.01475 (2020).

[41] J. Cohen, Statistical power analysis for the behavioral sciences. abingdon, England: Routledge (1988).

[42] S. Seabold, J. Perktold, statsmodels: Econometric and statistical modeling with python, in: 9^{th} Python in Science Conference, 2010.

[43] C. Colas, O. Sigaud, P.-Y. Oudeyer, How many random seeds? statistical power analysis in deep reinforcement learning experiments, arXiv preprint arXiv:1806.08295 (2018).

[44] K. Magnusson, Interpreting cohen's d effect size: An interactive visualization (2020).
URL https://rpsychologist.com/d3/cohend/

[45] L. Li, A. Talwalkar, Random search and reproducibility for neural architecture search, in: A. Globerson, R. Silva (Eds.), Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019, Vol. 115 of Proceedings of Machine Learning Research, AUAI Press, 2019, pp. 367–377.
URL http://proceedings.mlr.press/v115/li20c.html

[46] S. Xie, S. Hu, X. Wang, C. Liu, J. Shi, X. Liu, D. Lin, Understanding the wiring evolution in differentiable neural architecture search, arXiv e-prints (2020) arXiv:2009.01272arXiv:2009.01272.

[47] X. Dong, Y. Yang, Nas-bench-201: Extending the scope of reproducible neural architecture search, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.
URL https://openreview.net/forum?id=HJxyZkBKDr

[48] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, F. Hutter, NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search, arXiv e-prints (2020) arXiv:2008.09777arXiv:2008.09777.